Constrained Graphcut Texture Synthesis

Ganesh Ramanarayanan and Kavita Bala, Cornell University

***Abstract*— This paper describes constrained graphcut texture synthesis (CGS), a graphcut-based synthesis algorithm that creates output textures satisfying constraints. We show that constrained texture synthesis can be posed in a principled way as an optimization problem that requires balancing two measures of quality: constraint satisfaction and texture seamlessness. We then present an efficient algorithm for finding good solutions to this problem, using generalized graphcut minimization. CGS enables explicit control while preserving the speed and quality benefits of graphcut texture synthesis. This approach supports the full image analogies framework, while providing superior image quality and performance. A range of applications of CGS are demonstrated, including detail synthesis, artistic filtering by analogy, and texture-by-numbers. CGS is easily extended to handle multiple constraints on a single output, thus enabling novel applications that combine both user-specified and image-based control.**

***Index Terms*— texture synthesis, image analogies, detail synthesis, super-resolution**

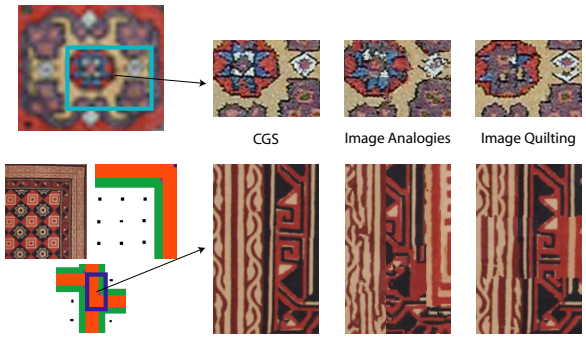


Fig. 1. Results showing detail synthesis (top) and user-controlled texture synthesis for texture-by-numbers (bottom). In both examples, CGS produces output superior to image analogies and image quilting. Top: carpet. The constraint is a blurry image. CGS is able to effectively match the constraint and synthesize plausible high-resolution detail. Bottom: cloth. Original input with its user specified correspondence are specified on the top. The constraint is shown in the bottom. The patch-based nature of CGS encourages coherence over the solid color regions of the constraint, and iterative refinement allows it to correct mistakes, resulting in properly placed, seamless patches in the output.

I. Introduction

Texture synthesis can decrease onerous modeling tasks by automatically creating textures from examples. Recent advances in texture synthesis technology [1] have dramatically improved both texture quality and synthesis performance. However, one problem with texture synthesis algorithms is that they are hard to control: they often fail in undesirable ways, producing unusable output. This has been an obstacle for the adoption of texture synthesis in settings where the user needs the output to have certain properties.

Several authors have recognized the benefit of controlling texture synthesis through analogy [2], [3]. The user sets up an "image analogy" by specifying image correspondences; in the notation of [2], these are called A, A', and B. The system then attempts to find an output B' that is related to the constraint B in the "same way" that A' is related to A. [2] uses pixel-based methods to compute this analogy, and applies the technique to a remarkable range of inputs, but it has some shortcomings. First, it is hard to precisely characterize what kind of image similarity the algorithm is aiming for. Second, the output quality suffers from the limitations of the underlying pixel-based synthesis algorithms. [3] achieve a similar "analogy" effect for the specific application of texture transfer by augmenting their image quilting algorithm with correspondence maps.

This paper presents **constrained graphcut texture synthesis** (CGS), a graphcut-based algorithm for texture synthesis that supports the image analogies framework in its full generality, while also leveraging the improved quality and performance of graphcut textures [1]. CGS supports the wide range of applications of image analogies, including detail synthesis, artistic filtering by analogy, and texture-by-numbers [2]. Constraints can be specified by a user, such as in texture-by-numbers, or through a constraint image, e.g. a low resolution image for detail synthesis. Additionally, multiple constraints can be combined for greater power.

In this work we make the following contributions. First, we give a principled formulation of constrained texture synthesis as an energy minimization problem. Second, we demonstrate how to approximate this energy minimization problem and solve it effeiciently using an adapted graphcut minimization algorithm. The key insight is that graphcut minimization can be used to make the output image agree with the imposed constraint while also seamlessly blending texture patches. Thus, constrained graphcut texture synthesis gives a new capability: a combination of explicit control (a la image analogies) with the performance and quality benefits of graphcut synthesis.

II. Related Work

Texture synthesis. Texture synthesis [4] focuses on the problem of using a small source texture to generate a large output texture that "looks the same". Pixel-based texture synthesis algorithms [5], [6] synthesize an output texture pixel-by-pixel, often using scale-space representations to match across different frequency bands. These approaches are particularly effective on stochastic textures, but they typically fail on textures with more coherent structure. Recent patch-based techniques [1], [3], [7] are better able to maintain visual coherence, and are fast, because they copy entire patches to the output instead of single pixels. However, most of these

algorithms are not able to constrain synthesis based on other images or user control.

Synthesis with constraints. General constraints to drive synthesis have been introduced in pixel-based [2], [8] and patch-based [3] contexts. [8] proposes user-drawn constraints as a guide for synthesis, and Image Analogies [2] presents a full framework supporting both user-drawn and image-based constraints. [9] uses a simplified version of the image analogies framework to generate textures from texton masks. The main benefits of CGS over these methods are the significant quality and performance gains of using patch-based graphcut synthesis instead of pixel-based.

In terms of patch-based techniques, Image Quilting [3] uses a correspondence map and hierarchical patch pasting to compute analogies, and [10] proposes an extension to [1] that is similar to our work. Both these papers demonstrate results for a particular image analogy application (texture transfer and artistic filtering, respectively). CGS achieves better quality and performance over the range of image analogies applications. See Sections III-E and IV for details.

Super-resolution and detail synthesis. Super-resolution and detail synthesis are slightly different approaches to the problem of enhancing a low-resolution image with high-resolution detail. Super-resolution [11] attempts to solve for the *actual* high frequency content in a low-resolution image. Common super-resolution algorithms take a low-resolution video sequence as input and extract additional data for one frame by analyzing its sequence of neighboring frames. On the other hand, detail synthesis [12] focuses on synthesizing *plausible* detail for a single low resolution image, using a small set of high resolution samples. [13] uses a Laplacian-like operation to split training images into low and high frequency, and then adds high frequency information to a low-resolution image in patches. The algorithm is promising for subtle edge enhancement at up to 4× linear (16× pixel), but it does not do as good a job of synthesizing plausible detail in contiguous regions. Detail synthesis for highly specialized problems using large training data sets has also been attempted for faces [14] and liver cells [15]. But detail synthesis is important for more general image-based rendering applications such as cultural heritage and surgical training, and furthermore, large training data sets are not always available. CGS could have a large impact here because of its ability to compute dramatic image zoom even with limited training data (up to 10× linear with a single source).

III. Constraint-Driven Texture Synthesis

We now describe our framework and algorithm for constrained patch-based texture synthesis. A principled approach is required to balance our two goals: satisfying constraints, and seamlessly synthesizing texture for the output. We formulate constrained texture synthesis as an optimization problem and describe how graphcut minimization can be used to balance these goals in a principled manner.

A. Framework

The CGS framework is depicted in Figure 2. The goal is to synthesize an output texture O, subject to the constraint C,

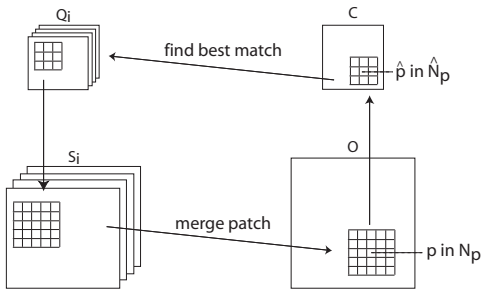


Fig. 2. Constrained patch-based texture synthesis overview.

using data from the sources $\{S_i\}$. In order to use $\{S_i\}$ in such a way that C is respected, there needs to be a way to compare neighborhoods of $\{S_i\}$ with neighborhoods of C, in order to determine what patches of $\{S_i\}$ are good matches for parts of C. However, in general, the domain of $\{S_i\}$ and the domain of C may be very different.

Let f be a function that maps the domain of $\{S_i\}$ to the domain of C. As input, CGS is given $\{S_i\}$, C, and a set of images $\{Q_i \mid Q_i = f(S_i)\}$, which we call the *mapped source* images. We can now compare neighborhoods of source S_k to a neighborhood of C by using the corresponding neighborhoods of Q_k. CGS then computes O such that C matches $f(O)$, and O consists of seamlessly composed patches from $\{S_i\}$. Thus, the Image Analogies relationship $A : A' :: B : B'$ is given here by $\{Q_i\} : \{S_i\} :: C : O$.

Our framework easily extends to handle multiple sources and mulitple constraints. In general, given sources $\{S_i\}$ and constraints $\{C_j\}$, and assuming some functions $\{f_j \mid f_j$ maps the domain of $\{S_i\}$ to the domain of $C_j\}$, we require as input mapped sources $\{Q_{ij} \mid Q_{ij} = f_j(S_i)\}$ to interrelate all sources and constraints. However, for the purposes of the following discussion, we will simply refer to a single constraint C, a single source S, and a single mapped source Q.

Constructing $f(O)$ for comparison against C might seem impossible since f is implicitly defined by Q. However, O is composed of patches from S and $Q = f(S)$, so when a patch from S appears in O, the corresponding patch from Q appears in $f(O)$. Thus, given O, we can compute $f(O)$ using this relation.

Here is an example of how to apply this framework. For detail synthesis, C is a low resolution image, S is a small, high-resolution example texture, and O is the image that matches C but has high frequency detail added. The function f used to create the Q would describe the downsampling and blur of the imaging device that captured C (Figure 2 shows 2× downsampling).

B. CGS as minimization

The goal of CGS is to compute a mapping from each pixel of the output O to a pixel in S. Let L be such a mapping. There are two separate measures of goodness for L: first, how closely $f(O)$ matches C, and second, how seamless the image O appears. Using this insight, CGS can be formulated as the

problem of choosing a mapping L that minimizes the energy function $E(L)$:

$$E(L) = K \sum_p \ A(p, L) + \sum_{(p,q)} \ M(p, q, L(p), L(q)) \quad (1)$$

The A function, the *agreement cost*, captures how well each synthesized pixel in O matches the constraint, and the M function, the *seam* cost, captures how seamlessly the patches blend with each other. The A and M functions are independent, and their relative weighting K must be a user input.

Our patch-based algorithm for minimizing this energy function is as follows (see Figure 2): (1) Find the output pixel p with the highest energy over its neighborhood. (2) Find the neighborhood from the input that best matches the constraint corresponding to p (at $\hat{p}$). (3) Merge the patch containing that neighborhood into the output seamlessly by adjusting the patch boundary to minimize energy. (4) Repeat until the termination condition is satisfied. We describe each step of the algorithm in detail in Section III-D.

Phrasing CGS as an energy minimization problem makes it easy to support multiple constraint images, each with its own cost function. For example, we combine an image-based constraint with a texture-by-numbers constraint to perform linear $10\times$ zoom for detail synthesis (Figure 3B)

C. Energy minimization using graph cuts

A brute-force implementation of some of the steps in the above algorithm would be impractically expensive. For example, step 3 requires minimizing over all possible ways to trim a patch. Our insight is that graphcut minimization [16], a popular computer vision technique, can be extended beyond [1] to minimize this energy function. The challenge is to approximate the energy function (Equation 1) in a way that permits use of graphcut minimization.

Graph cut review: Graphcut minimization can be used to assign labels to a grid of pixels such that the assignment minimizes an energy function of the form $E(L) = \sum_p \ U(p, L(p)) + \sum_{(p,q)} \ V(p, q, L(p), L(q))$, where p is a pixel, (p, q) are neighboring pixels, $L(p)$ is the label assigned to p, U is the *assignment cost* and V is the *separation cost*. For vision problems, labels typically represent disparity or depth. The algorithm consists of repeated α*-expansion* moves. An α-expansion move permits every pixel to either keep its label or flip it to alpha, and it picks the flip that reduces energy the most. In the full minimization algorithm, expansion is performed repeatedly for all labels α until convergence. The optimality of the solution obtained by this algorithm depends solely on V. As V gets more complicated, optimality guarantees weaken, but results are still excellent in practice.

Graphcut textures [1] introduced an iterative patch-based texture synthesis algorithm that only used the V cost in graphcut minimization to find the best seam between patches. The assignment cost U is unused in their formulation. We show how to use U to achieve our goal of constrained graphcut texture synthesis.

Graph cuts for CGS: To match constraints, CGS introduces the agreement cost A. Our insight is that A corresponds roughly to the U term in graphcut minimization. However, the correspondence is not exact; an approximation of E is needed to use graphcut minimization for CGS.

The graphcut framework requires labels and a function to be optimized. We use a labeling scheme similar to entire patch matching [1], where a label corresponds to different translations and rotations of the input images $\{S_i\}$. Thus, given an output pixel p, the label $L(p)$ identifies the corresponding input pixel. Contiguous regions of a single label correspond to seamless copying. We define $g(L, p)$ to be the function that looks up the appropriate pixel from S to produce the output pixel $O(p)$. For a given pixel p in O or S, we refer to its corresponding pixel in C or Q as $\hat{p}$. We denote a neighborhood of pixels in O or S around the pixel p as N_p. We denote the corresponding neighborhood of pixels in C or Q around $\hat{p}$ as $\hat{N}_p$.

Since the function A captures how well the output matches the constraint, it is possible to define it as a pixel-by-pixel comparison between the output (actually, $f(O)$) and the constraint. However, using neighborhoods in this comparison is much better because neighborhoods capture visual coherence and continuity. Therefore, we define A as:

$$A(p, L) = K \ SSD_{\hat{N}_p}(f(g(L)), C) = K \ SSD_{\hat{N}_p}(f(O), C) \quad (2)$$

where $g(L) = O$, and $SSD_N(a, b)$ is the sum of square differences between images a and b over a neighborhood N: $\sum_{p \in N} |a_p - b_p|^2$. K is again the user-specified parameter that controls the relative weighting of the agreement and seam costs.

The seam cost M imposes a penalty for changing labels. It is defined in the same way as in graphcut textures: $M(p, q, L(p), L(q)) = ||P_p(p) - P_q(p)||_2 + ||P_p(q) - P_q(q)||_2$, where P_p and P_q are the patches associated with labels $L(p)$ and $L(q)$.

The assignment cost U is nearly suited to represent A. However, the definition of A is not directly applicable in graphcuts because it is defined over L, rather than for a single label (as required by the graphcut framework). Therefore, we approximate A with another function A' that does fit into the graphcut framework:

$$A'(p, l) = K \ SSD_{\hat{N}_p}(f(P_l), C)$$

Here l is equal to $L(p)$ and P_l is the source patch corresponding to label l. Note that if patch P_l is pasted over all of N_p in O, then $A(p, L) = A'(p, l)$. Intuitively, this approximation treats a neighborhood of pixels with possibly different labels as if it were a neighborhood of pixels all with label l. In general, $A \neq A'$, because the neighborhood may contain seams with different labels on the other side. The reason the A' approximation works well is that optimizing the seam cost M tends to make the actual pixels on the other side of the seam agree with the pixels from the neighborhood P_l.

This approximate energy function is then amenable to graphcut minimization:

$$E'(L) = \sum_p \ A'(p, L(p)) + \sum_{(p,q)} \ M(p, q, L(p), L(q)) \quad (3)$$

Many labels. CGS presents a unique challenge for graphcut minimization because the set of labels is large. In a constrained framework, the set of valid labels could easily include every affine transformation of a patch. Translations alone require $O(n)$ labels, where n is the number of pixels in the output image; adding even basic rotations expands this further. Because of the lack of constraints, [1] does not need to paste many patches to produce decent output (on the order of tens). By contrast, CGS needs to adapt to all the features in the constraint image, and thus it needs to paste many more times (50-100 in our examples). Furthermore, to get the best output, one needs to consider as much of the full label space as possible. Even for small images (e.g. $n = 400 \times 400$) the space is huge.

Large label sets remain an open problem in graphcut minimization because, in general, the minimization algorithm requires that all labels be considered for α-expansions, and with a large label set, this linear dependence on the number of labels is prohibitively expensive. We apply a simple heuristic: the algorithm picks a small set of m candidate labels for α-expansions (typically $m = 10$). Experimentally, output quality seems to be insensitive to increasing this parameter. We avoid picking tightly clustered labels by enforcing that each label lies beyond some minimum cutoff distance from any other candidate label (like the Poisson distribution in [17]).

D. Detailed Algorithm

We now describe the algorithm in detail (see Figure 2).

Step 1: Output refinement location. Find the *worst* output pixel p for refinement by iterating over all pixels in the output and approximating $E = A + M$ from Equation 2. Our approximation is to evaluate the A cost only at pixel p and the M cost over the entire neighborhood. Pick the pixel p with the highest total energy.

Step 2: Patch finding. Find the corresponding constraint pixel $\hat{p}$. Select candidate patches from the source images that both closely match the constraint for pixel $\hat{p}$, and blend well with the output. These candidates are selected by computing an approximate agreement and seam cost for all patches. A given candidate patch in S corresponds to a label l, and is denoted as P_l. The corresponding patch in Q_i is denoted as Q_l. The agreement cost for l is approximated by an SSD between neighborhoods of $\hat{p}$ in C and Q_l. The seam cost is approximated by an SSD between the neighborhoods for p in O and P_l.

Thus, the best candidate patch corresponds to label l: $argmin_l \; K \; SSD_{\hat{N}_p}(Q_l, C) + SSD_{N_p}(P_l, O)$. However, as described in Section III-C, the best m labels (not just one) are found to be considered as α-expansions. To rapidly find the best match over all possible input translations we use summed area tables and convolution as in [1].

Step 3: Patch merging. Once the prospective m patches are selected, instantiate graph cuts with $U = A'$, $V = M$ for all pixels in the new patches and try all m α-expansions. Graphcuts will automatically find the best seam between each new patch and the existing patch. Pick the best expansion, i.e., the highest drop in energy among the m patches, as the final patch to be copied into the output.

Step 4: Termination. On each iteration, the algorithm maintains a history of h α-expansions that have been attempted by graph cuts at different pixels ($h = 15$ for all our examples). If the h expansion moves do not decrease error by more than some tolerance t, iterative synthesis is terminated. Because energy increases with K, t should be proportional to K. We use $t = 10(K + 2)$ for all our examples; smaller values of t do not appear to affect image quality.

E. Algorithmic comparison with other patch-based techniques

As mentioned in the related work, CGS bears the most similarity algorithmically to [3] and [10]. We will now describe these algorithms in some detail and contrast their approaches with ours.

Image Quilting [3]. The constrained version of this algorithm is a multi-pass hierarchical technique. First, patches of a certain size are selected from $\{S_i\}$ (say 64×64) and pasted into O in raster order, using the quilting technique. Patches are selected based on how well they match C and the existing output in O. Call this output O_{prev}. Then, in the next pass, the patch size is reduced, and the same synthesis algorithm is repeated, this time incorporating match cost with O_{prev} into the patch selection, and also increasing the weight of C. O_{prev} is updated, and this iterates several times. To fully match the constraint, it is important to use very small patches in the final pass.

[3] only demonstrates results for texture transfer, but detail synthesis results have reasonable quality as well. However, this algorithm has two shortcomings. One is that the artifacts of the raster-scan patch quilting approach are often evident in the output, in the form of sharp vertical / horizontal seams or block-like behavior. The other is performance. For example, given a 500×500 image, and a final patch size of 7×7, one needs to perform 6889 patch searches just for the final pass. CGS performs around 50-100 patch searches for the full algorithm. Thus, our running time for such an example is on the order of 10 -15 minutes, while the image quilting algorithm can take upwards of 12 hours.

Schödl Ph.D. Thesis [10] This algorithm is a proposed extension to [1], where an assignment cost is added to the graph cut optimization, and patch placement is totally random. In this case, the only demonstrated results are for artistic filtering, and the random placement algorithm works sufficiently well for simple cases (see supplemental images). However, as shown in Figure 3A, this technique is not as effective for solving problems like detail synthesis. (see submitted images for further comparisons). Even if the algorithm is run for an hour or more, trying more than 20000 expansion moves, this technique is unable to match the quality of CGS. As mentioned in our discussion of many labels, introducing an assignment cost significantly changes the behavior of the optimization framework. It is difficult for random placement to adapt quickly and effectively to the features of the constraint image, especially when both the source and constraint are highly structured.

IV. RESULTS

We show results of our CGS algorithm and compare with Image Analogies [2] (IA), Image Quilting [3] (IQ), and Schödl [10] in Figures 3 and 4. We encourage the reader to look at the full resolution submitted images when reviewing the results.

Performance. We used the publicly available version of IA from NYU's Media Research Lab, and our own implementation of IQ. All results were obtained on a Pentium Xeon 3.6 GHz machine with 2 GB RAM. The running times for CGS ranged from 5 to 15 minutes for all examples shown in the paper. By comparison, on the same machine, the running time ranges for IA and IQ were 15 minutes - 1.5 hours and 40 minutes - 16 hours, respectively. Typically, larger detail synthesis outputs take the most time for all of these algorithms. Schödl's running time is dependent on the number of iterations chosen by the user, so for fair comparisons to CGS we ran Schödl for a comparable amount of time.

Detail synthesis. We demonstrate the ability of CGS to synthesize detail in Figure 3A, where CGS is applied to a blurred carpet (3A-(iii). For comparison, we used the IA, IQ, and Schödl algorithms to generate similar results (Figure 3A-(iv-vi)). The original constraint image is much larger; only 2 medallions are shown (see submitted images). The IA output is much better than that published in their original paper, because we provided better training data and tweaked some parameters. Even so, CGS performs better than IA and the other techniques as well. Schöodl's algorithm performs particularly poorly on the bottom carpet medallion, and IQ has difficulty matching a medallion that is not exactly the same shape as the medallions in the source (see zoom-in).

In Figure 3B we demonstrate the use of multiple constraints for a $10\times$ linear magnification of brick (i.e., $|O| = 100|C|$). The low-resolution brick wall provides one constriant, and the black lines provide a second constraint guiding where the grout should be. CGS maintains the brick color variations and small perturbations in grout and brick shadows, while also keeping the brick/grout edges reasonably straight, as dictated by the second constraint. Figure 3C shows $3\times$ linear zoom of woven cloth. It correctly adds weave detail to the high resolution output while matching the constraint, including the small red spot in the leaf/stalk pattern.

We now show other applications from [2], [3]. **Artistic filters.** For the Freud painting example we first use the simple luminance matching technique of [2] to bring the S and Q color spaces into correspondence with C (Figure 4B). We show outputs of the IA, IQ, and Schödl algorithms. The results are comparable, but CGS better preserves details like the thin trees in the center and right, and image quilting suffers from some repetitive block copying.

Texture transfer. Texture transfer involves a tension between matching the constraint and generating convincing texture. Figure 4C shows that CGS can match the constraint well and generate a nicer weave texture than IA. In particular, the region around the woman's nose is noticeably better. IQ produces a nice weave texture as well, but it is unable to capture the nose properly. Figure 4D shows the rice texture transfer example from [3]. CGS does less repetitive copying and generates better seams than IQ in the mouth region.

Texture-by-numbers. Some texture-by-numbers examples are shown in Figures 4A, 4E, and 4. The cloth example (A) benefits greatly from patch-based techniques. CGS handles the distortion of the cloth border by synthesizing the confluence of the line patterns; IA produces a much less coherent texture. IQ performs better than IA, but due to the sensitivity of its patch size parameter, it is unable to capture all structures in the output seamlessly. In Figure 4E, CGS avoids synthesizing unnatural streaks in the water that arise in the IA output from "garbage" growing. It also captures the inner contours of the arch better than IQ, which demonstrates noticeable block artifacts from raster-scan square patch refinement.

4F shows a pathological case for CGS. The source image does not have many dense blue regions, so CGS has great difficulty finding large neighborhoods that match the curved constraint symbol well, resulting in patchy artifacts in the output. In general, pixel-based methods such as IA will have greater agility in adapting to subtle pixel-level changes when compared to patch-based methods such as CGS.

Discussion: We now discuss strengths and weaknesses of CGS and compare patch-based and pixel-based synthesis with constraints.

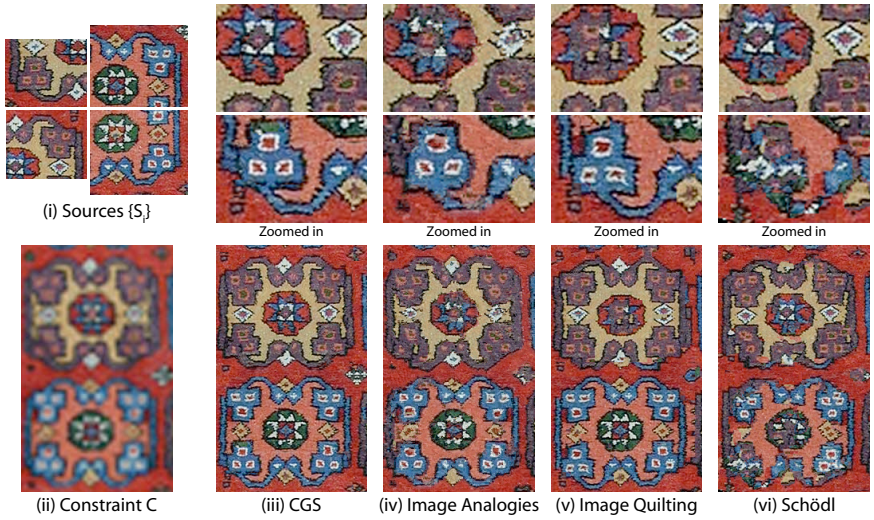
Correspondence between Q_i and C: Both CGS and analogies require some correspondence between Q_i and C; otherwise, CGS effectively degenerates to texture transfer. Therefore, for artistic filters, it is hard to define success. When Q_i and C lack significant similarity, M will have poor properties, often violating *regularity* [18]. If M is not regular, ordinary graph cut minimization can fail; we have implemented a correction [19] to avoid this pitfall (the details are beyond the scope of this paper).

Patch size: SSDs are typically computed over small neighborhoods (from 5 to 15 pixels), instead of the full patch, to improve the chance of locally matching C. However, large patches are better for patch merging because they give α-expansions more flexibility. Therefore the entire source image is used for patch merging.

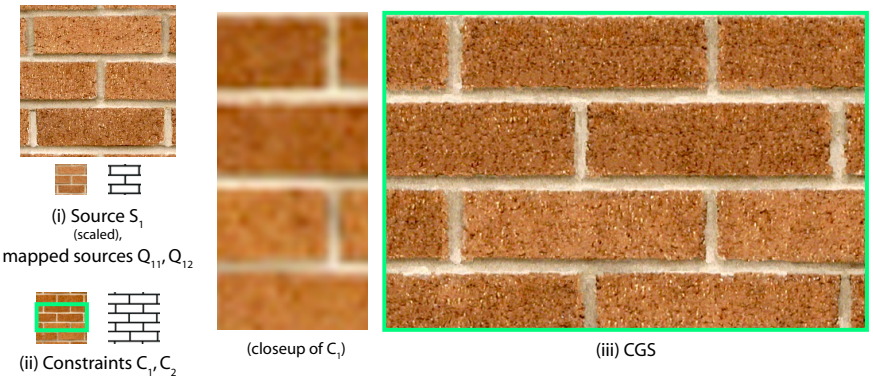
Synthesis parameters: Using sophisticated energy minimization makes CGS more robust; smoothly changing parameter settings produce smoothly changing output characteristics. The most important parameters are N, the neighborhood size, and K, the relative weighting of agreement and seam costs. Varying N controls how much local information is taken into account; it is set higher in detail synthesis when good matches are important. For underconstrained synthesis, such as texture-by-numbers, K should be small because the constraint consists mostly of smooth gradients or solid color regions, for which there will typically be many matches.

Tradeoffs: Patches do a very good job of preserving local coherence, but pixels are better at adapting to radical changes in structure (melody example in submitted images), which in general patch-based methods do not handle well. [20] and [21] introduce warping, but heavy deformation causes unwanted blurring. This is an area for future work.

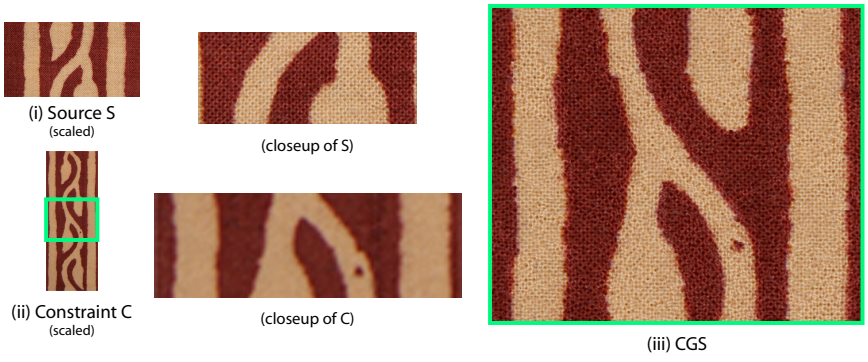
Good examples are needed: If the examples are poorly chosen so a match cannot be found in the constraint, the



A. Detail Synthesis: Reconstruction of carpet image



B. Detail synthesis: Brick 10x linear zoom with multiple constraints



C. Detail Synthesis:: Cloth 3x linear zoom

Fig. 3. CGS for detail synthesis, run with $N = 15$, $K = 8$. In all figures, (i) is source S, (ii) constraint C, (iii) CGS output, and, if present, (iv) image analogies output, (v) image quilting output, (vi) Schödl output. (A) Carpet image is restored from a blurry constraint. (B) $10\times$ linear magnification of brick with multiple constraints. Close examination shows how well the output matches subtle color and shadow changes in the low resolution image. (3) $3\times$ linear magnification of cloth weave pattern (part of the output shown). CGS is able to adapt to variability in the leaf shapes.

algorithm will blend patches that are poor matches, as is to be expected.

V. Conclusions

Constrained texture synthesis, such as image analogies, is a powerful capability with many applications. This work shows how to do graphcut texture synthesis that respects constraints and yields high-quality results. Therefore it should have significant impact on the use of image analogies in practice.

We make the following contributions: we formulate constrained texture synthesis in a principled way as a minimiza-

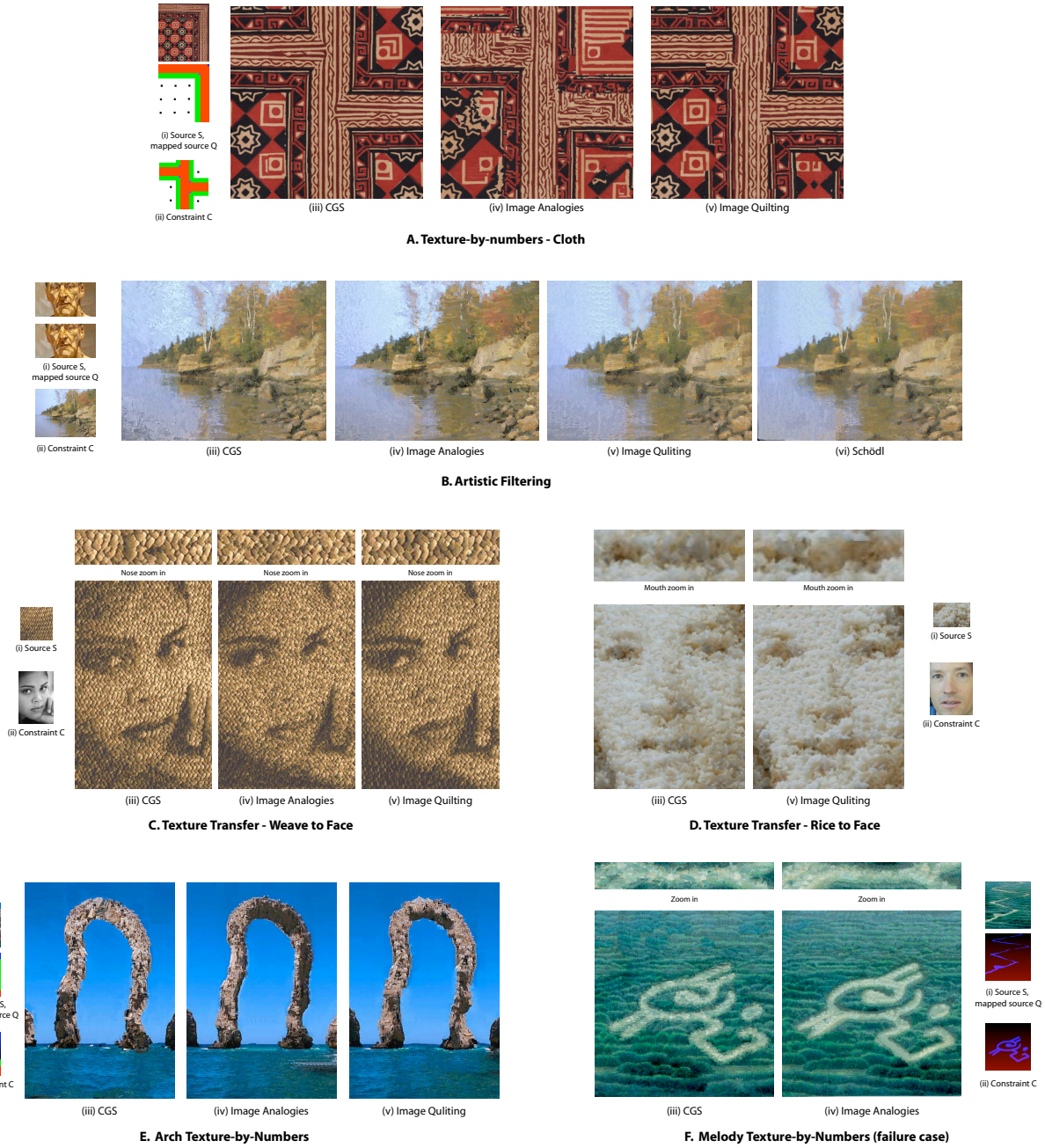


Fig. 4. Artistic examples (parameters in parenthesis) (A) Cloth texture-by-numbers. ($K = 2.0$, $N = 11$) CGS produces good output, while IA is under-constrained and synthesizes incorrectly. IQ's fixed patch size makes it hard to capture large and small structures in a seamless texture. (B) Artistic filter. ($K = 20.0$, $N = 5$) Results are comparable, but CGS matches fine details like the thin trees better. (C) Weave texture transfer. ($K = 1.0$, $N = 5$) CGS is able to both match the constraint and synthesize a coherent weave texture. Note the comparisons of the nose. (D) Rice texture transfer ($K = 0.2$, $N = 5$). Again, CGS matches the constraint closely while finding better patch seams. (E) Arch texture-by-numbers. ($K = 2.0$, $N = 5$) CGS avoids synthesizing streaks in the water (improvement over IA) and adapts to the smooth inner contours of the arch without blocky artifacts (improvement over IQ) (F) Melody texture-by-numbers. ($K = 2.0$, $N = 5$) Failure example. CGS does not have the agility to match the blue symbol constraint well, given that the blue region in the mapped source is thin and irregularly shaped.

tion problem that requires simultaneous optimization of both the agreement between the output and constraint image, and the seamlessness of the patches making up the output. We show how to approximate this energy minimization so that a graph cut minimization algorithm can be used to efficiently find solutions.

We have demonstrated CPS on a range of applications: detail synthesis, texture-by-numbers, artistic filters, and other image analogies. The output has better quality and performance compared to other pixel-based and patch-based analogy approaches. Results for detail synthesis are particularly good, especially with the use of multiple constraints.

In future work, more general patch and color transformations would extend the power of this technique. Another interesting avenue is using the energy minimization framework to implement perceptual measures of texture quality.

REFERENCES

[1] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," in *SIGGRAPH '03*, 2003, pp. 277–286.

[2] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *SIGGRAPH '01*, 2001, pp. 327–340.

[3] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *SIGGRAPH '01*, 2001, pp. 341–346.

[4] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *SIGGRAPH '95*, 1995, pp. 229–238.

[5] J. S. D. Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images," in *SIGGRAPH '97*, 1997, pp. 361–368.

[6] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *SIGGRAPH '00*, 2000, pp. 479–488.

[7] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, 2001.

[8] M. Ashikhmin, "Synthesizing natural textures," in *Symposium on Interactive 3D Graphics*, 2001, pp. 217–226.

[9] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum, "Synthesis of progressively-variant textures on arbitrary surfaces," in *SIGGRAPH '03*, 2003.

[10] A. Schödl, "Multi-dimensional exemplar-based texture synthesis," Ph.D. dissertation, Georgia Institute of Technology, 2002.

[11] S. Borman and R. L. Stevenson, "Super-resolution from image sequences - A review," in *Proceedings of the 1998 Midwest Symposium on Circuits and Systems*, Notre Dame, IN, 1998.

[12] R. Ismert, K. Bala, and D. Greenberg, "Detail synthesis for image-based texturing," in *I3D'03*, Apr. 2003, pp. 171–176.

[13] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Comput. Graph. Appl.*, vol. 22, no. 2, pp. 56–65, 2002.

[14] C. Liu, H.-Y. Shum, and C.-S. Zhang, "A two-step approach to hallucinating faces: global parameteric model and local nonparametric model," in *CVPR*, 2001, pp. 192–198.

[15] L. Wang and K. Mueller, "Generating sub-resolution detail in images and volumes using constrained texture synthesis," in *Proceedings of the IEEE Visualization 2004 (VIS'04)*, 2004, pp. 75–82.

[16] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, 2001.

[17] S. Zelinka and M. Garland, "Jump map-based interactive texture synthesis," *ACM Transactions on Graphics*, vol. 23, no. 4, pp. 930–962, 2004.

[18] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?" *IEEE PAMI*, vol. 26, no. 2, pp. 147–159, 2004.

[19] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake, "Digital tapestry," in *submitted to CVPR*, 2005.

[20] Q. Wu and Y. Yu, "Feature matching and deformation for texture synthesis," in *SIGGRAPH '04*, 2004, pp. 364–367.

[21] Y. Liu, W.-C. Lin, and J. Hays, "Near-regular texture analysis and manipulation," *SIGGRAPH '04*, pp. 368–376, 2004.